CSC 317: Data Structures and Algorithm Analysis

Dilip Sarkar

Department of Computer Science
University of Miami

Outline I

Hiring problem

- **Problem:** Suppose you need an assistant to help with your daily schedule. You have found an employment agency who will provide you n candidates, one every day. You want to hire **the best candidate**.
- **Availability of candidates:** After an interview, if you do not hire a candidate, s/he is not available any more.
- **Hiring and firing cost:** To interview a candidate cost you c_i and hiring a candidate cost you c_h, where $c_i << c_h$.
- **Total cost**: If you hire m candidates, your total cost is $nc_i + mc_h$.
- **Algorithm:** If new candidate is better than current assistant, fire current assistant and hire the new candidate.

Expected time complexity of Quick Sort, Selection, and hash tables
Selection in expected linear time

Hiring problem (cont.)

- **Problem:** Suppose you need an assistant to help with your daily schedule. You have found an employment agency who will provide you n candidates, one every day. You want to hire **the best candidate**.
- **Availability of candidates:** After an interview, if you do not hire a candidate, s/he is not available any more.
- **Hiring and firing cost:** To interview a candidate cost you c_i and hiring a candidate cost you c_h, where $c_i << c_h$.
- **Total cost**: If you hire m candidates, your total cost is $nc_i + mc_h$.
- **Algorithm:** If a new candidate is better than current assistant, fire current assistant and hire the new candidate.

```
HIRE-ASSISTANT(n)
   best = 0 { candidate 0 is the lest-qualified
              dummy candidate }
  for i = 1 to n
      interview candidate i
      if candidate i is better than
            candidate best
          best = i
          hire candidate i
```

- **What is the hiring cost?**
- **Best case:** $nc_i + c_h$
- **Worst case:** $nc_i + nc_h = n(c_i + c_h)$
- **Expected cost:** Depends on the order in which candidates arrive.
- **Probabilistic case:** use probability of hiring a candidate for estimating **expected cost**.
- **Randomized algorithm:** We randomize the input.
- **Randomization hiring problem:** Get a list of n candidates, and select candidate for interview in a random order.

Indicator random variables

- Sample space: S;
- Examples
  - Coin toss: $S = \{H, T\}$
  - Roll of a (six-sided) dice: $S = \{1, 2, 3, 4, 5, 6\}$
- An event: A
- Example
  - Coin toss: outcome is a head $\Rightarrow A \in \{H\}$
  - Roll of a (six-sided) dice: out come is an even number $\Rightarrow A \in \{2, 4, 6\}$
- Indicator random variable $X_A = I\{A\}$ associated with A
  - $X_A = I\{A\} = 1$ if A occurs
  - $X_A = I\{A\} = 0$ if A does not occur
- Example: From toss of a coin $A = H$
  - We have $X_A = X_H = I\{H\} = 1$, if H occurs
  - We have $X_A = X_H = I\{H\} = 0$, if T occurs
- Expected value of a random a random indicator variable X_A
- Expected number of heads from a fair-coin toss, X_H
  - $E[X_H] = E[I\{H\}] = 1{\cdot}\Pr\{H\} + 0{\cdot}\Pr\{T\}$
    $= 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2$
- *Lemma 5.1*
  Given a sample space S and an event A in the sample space S, let $X_A = I\{A\}$. Then $E[X_A] = \Pr\{A\}$.
- *Proof:*
  By definition of X_A,
  $E[X_A] = E[I\{A\}] = 1{\cdot}\Pr\{A\} + 0{\cdot}\Pr\{\overline{A}\}$
  $= 1{\cdot}\Pr\{A\} + 0{\cdot}\Pr\{\overline{A}\} = \Pr\{A\}$

Some applications of indicator random variables

Example 1:

- Fair and unfair coins.
  - Sample space of a fair coin: $S = \{H, T\}$
  - If a fair coin is tossed, $\Pr(H) = P(T) = 1/2$.
  - An extreme example of an unfair coin
  - Both sides has tail and thus, sample space: $S = \{T\}$. $\Pr(T) = 1$ and $\Pr(H) = 0$.
- Let a bag has 25 coins. Twenty of them are **fair** and five are **unfair** — both sides have tails. All 25 coins in the bag are tossed together.
- Let $X_i = I\{$toss of the coin i is head $H\}$, that is, $X_i = 1$ if toss of the coin i is head, else $X_i = 0$. $E[X_i] =$Pr(toss of coin i is head.)
- $E[X_i] = 1/2$ for a fair coin. $E[X_i] = 0$ for an unfair coin.
- Problems
  1. How many ways we can get **exactly one** head?
  2. 20, only one of the 20 fair coins shows a head.
  3. Mathematically, $\binom{20}{1} = 20$
  4. How many ways we can get **exactly two** heads?
  5. 190, because $\binom{20}{2} = \frac{20 \cdot 19}{2 \cdot 1} = 190$
- Let random variable
  $X = X_1 + X_2 + \cdots + X_{25} = \sum_{i=1}^{25} X_i$.
  1. What is the range of values for X?
  2. Zero if none of coins shows a head
  3. Maximum number of heads are 20, because 5 unfair coins will never contribute to the sum.
- What is expected (average) number of heads $E[X]$, if we toss all 25 coins?
- $E[X] = E[\sum_{i=1}^{25} X_i] = \sum_{i=1}^{25} E[X_i]$, because (expectation of sum) = (sum of expectations).
- We need to divide the sum into two parts: for 20 fair coins and 5 unfair coins.
- $E[X] = E[\sum_{i=1}^{20} X_i] + \sum_{j=1}^{5} E[X_j]$

  $= 20 \cdot \frac{1}{2} + 5 \cdot 0 = 10$
- Okay! Quite a bit abstract notations and simple math. BUT
- **What this has to do with algorithm analysis?**

Analysis of hiring algorithm

- Let X_i be the indicator random variable associated with the event that candidate i is hired.
  - $X_i = I\{X_i\} = 1$, if candidate i is hired
  - $X_i = I\{X_i\} = 0$, if candidate i is not hired
- Let $X = X_1 + X_2 + \cdots + X_n = \sum_1^n X_i$
- $E[X] = E(\sum_{i=1}^n X_i) = \sum_{i=1}^n E[X_i]$,
  by distribution of sum of expectations
- What is the value of $E(X_i)$?
- $E(X_i) =$Pr{candidate i is hired}
  [by Lemma 5.1]
- What is the probability that the candidate i is hired?
- Pr{candidate i is hired} $= 1/i$.
- Why $1/i$?
- Thus, $E[X] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n (1/i) = \ln n + O(1)$

```
HIRE-ASSISTANT(n)
best = 0 { candidate 0 is the lest-qualified
           dummy candidate }
for i = 1 to n
    interview candidate i
    if candidate i is better than
           candidate best
        best = i
        hire candidate i
```

- Okay!
- BUT, this algorithm is for an artificial problem and not solving a **real** Computer Science problem.
- **What this has to do with algorithm analysis?**

Complexity of quick sort algorithm

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

- $T(n) = T(k-1) + T(n-k) + cn$
- Worst cases
  - $(k-1) = 0$, remaining elements are in the right partition
  - $(k-1) = n-1$, remaining elements are in the left partition
  - In both cases $T(n) = O(n^2)$
- Best cases 1
  - $(k-1) \cong n/2$, partitions are always almost equal
  - In this case $T(n) = O(n \lg n)$
- Best cases 2
  - $(k-1) \cong (1/c_1)(n/2)$,
  - Ratio of lengths of two partitions is a constant c_1 (or $1/c_1$)
  - In this case $T(n) = O(n \lg n)$
- What is expected running time $E[T(n))]$?
- To compute $E[T(n))]$, we have to consider randomized quicksort.
- We will show that $E[T(n))] = O(n \lg n)$
- First we consider randomized partitioning.

Probability of comparing two elements during partition

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

- Suppose we call RANDOMIZED-PARTITION$(A, 1, k)$
- For $1 \le i < j \le k$, will **A[i]** be compared with **A[j]**?
- Yes, if **A[i]** is the pivot or **A[j]** is the pivot.
- What is the probability that **A[i]** is selected as the pivot?
- If the array has k elements, then $(1/k)$.
- Similarly, probability that **A[j]** is selected as the pivot is $(1/k)$.
- Probability that **A[i]** or **A[j]** selected as pivot is $((1/k) + (1/k)) = (2/k)$.

  $\Rightarrow$ probability that **A[i]** is compared with **A[j]** is $(2/k)$.

Prob. of comparing two elements during partition (cont.)

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q − 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

- Suppose we call RANDOMIZED-QUICKSORT$(A, 1, n)$ to sort an array of n elements.
- How many times the procedure RANDOMIZED-PARTITION(A, p, r) is called?
- RANDOMIZED-PARTITION(A, p, r) is called at most n times.
- Excluding comparisons, each call to RANDOMIZED-PARTITION(A, p, r) takes $O(1)$ time.
- How many times two elements **A[i]** and **A[j]** are compared?
- At most once. Why?
- Because if **A[i]** is compared with **A[j]**, then either **A[i]** or **A[j]** was the pivot. Also, after they are compared, they will never be in the same partition again.

Expected running time of randomized Quick sort

Now let us put everything together

- For $p \leq i < j \leq r$, and we call RANDOMIZED-PARTITION(A, p, r), let
  $X_{pr} = I\{$ **A[i]** is compared with **A[j]**$\}$
  $=\Pr\{$**A[i]** is compared with **A[j]**$\}$
  $= \frac{2}{(r-p+1)}$
- Let X be the total number of comparisons
-
$$X = \sum_{p=1}^{n-1} \sum_{r=p+1}^{n} X_{pr}$$
-
$$E[X] = \sum_{p=1}^{n-1} \sum_{r=p+1}^{n} E[X_{pr}]$$
$$= \sum_{p=1}^{n-1} \sum_{r=p+1}^{n} \frac{2}{(r-p+1)}$$

- Now if $(r - p) = k$ we have,
$$E[X] = \sum_{p=1}^{n-1} \sum_{k=1}^{n-p} \frac{2}{(k+1)}$$
$$< \sum_{p=1}^{n-1} \sum_{k=1}^{n} \frac{2}{(k+1)}$$
$$= \sum_{p=1}^{n-1} O(\lg n)$$
$$= O(n \lg n)$$

Steps of Randomized Partition Algorithm

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]
q = RANDOMIZED-PARTITION(A, p, r)
k = q - p + 1
if i == k            // the pivot value is the answer
    return A[q]
elseif i < k
    return RANDOMIZED-SELECT(A, p, q - 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- The 1st procedure on the left is a randomized selection algorithm
- It uses a randomized partition procedure, RANDOMIZED-PARTITION(A, p, r)
- The procedure RANDOMIZED-PARTITION(A, p, r) is the middle procedure
- Procedure RANDOMIZED-PARTITION(A, p, r)
  - Line 1: A random integer generator RANDOM(A, p, r) is called to generate i, such that $p \leq i \leq r$
  - Line 2: $A[i]$ is exchanged with $A[r]$
  - Line 3: The elements in $A[p \cdot\cdot r]$ is partitioned using $A[r]$ as the pivot
  - NOTE: when PARTITION(A, p, r) is called, $A[r]$ is $A[i]$ of the original $A[p \cdot\cdot r]$ before the exchange of $A[r]$ with $A[i]$ on Line 2.

Randomized Selection Algorithm

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2      return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k          // the pivot value is the answer
6      return A[q]
7  elseif i < k
8      return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

- The first call: $\textsc{Randomized-Select}(A, 1, n, i)$; i is the rank of the element we want to find.
- Line 1: if $p = r$, we have only one element and the p is the index of the ith element
  - This line is unlikely for the FIRST call
  - Because then $i = 1$ and randomized algorithm would make no sense.
- Line 2: return $A[p]$
- Line 3: A random integer q, such that $p \leq q \leq r$, is generated
  - Procedure $\textsc{Randomized-Partition}(A, p, r)$ is called for generating q;
- Line 4: After partition is complete, the **rank** k of the pivot is calculated
- Line 5: If rank $k = i$, we have found the element we are looking for
- Line 6: **return** $A[q]$
- (Lines 7 and 8) **OR** (Lines 7 and 9) are executed when $A[q]$ is NOT the element being searched.
- Line 7: Is ith element on the left of $A[q]$, the kth element?
  - YES $\Rightarrow$ Line 8: Call $\textsc{Randomized-Select}(A, p, q-1, i)$ with partition to the left of $A[q]$
  - NO $\Rightarrow$ Line 9: Call $\textsc{Randomized-Select}(A, q+1, r, k-i)$ with partition to the right of $A[q]$

Expected Performance of Randomized Selection Algorithm


```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2      return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k          // the pivot value is the answer
6      return A[q]
7  elseif i < k
8      return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```


- Worst-case time complexity: Pivot is always the minimum or the maximum of the array under consideration
  - $T(n) = T(0) + T(n-1) + O(n)$
  - Solution to the above equation $T(n) = O(n^2)$
- Now we compute an estimate of $T(n)$
- Before we proceed further, note that $1 \leq q \leq n$
- When $q = 1$, we have only one partition, $(A[2 \cdots n])$.
- Similarly, when $q = n$, we have only one partition, $(A[1 \cdots (n-1)])$.
- For cases $2 \leq q \leq (n-1)$ the array is divided into two partitions.
- $(A[1 \cdot\cdot 1], A[3, \cdots n]), (A[1 \cdot\cdot 2], A[4, \cdots n]), \cdots, (A[1 \cdot\cdot (n-2)], A[n, \cdots n])$
- For $2 \leq q \leq (n-1)$, we do not know which subarray will be used.
- To establish an upper bound we will consider the larger subarray of the two, that is, $\max\{(k, n-(k+1))\}$ for $1 \leq k \leq (n-2)$.
- The length of subarrays need consideration are: $(n-1), (n-2), (n-3), \cdots, \lfloor (n/2) \rfloor$, $(\lfloor (n/2) \rfloor + 1), \cdots, (n-3), (n-2), (n-1)$
- For even n, each value is occurring twice. For odd n we include an extra value of, $\lfloor (n/2) \rfloor$
- Since q is selected from an array of n elements, $\Pr(q \in \{1, 2, \cdots n\}) = 1/n$

Expected Performance of Randomized Selection (II)

- Last four lines from previous slide
  - To establish an upper bound we will consider the larger subarray of the two, that is, $\max\{(k, n-(k+1))\}$ for $1 \leq k \leq (n-2)$.
  - The length of arrays need to be considered are: $(n-1), (n-2), (n-3), \cdots, \lfloor (n/2) \rfloor$, $(\lfloor (n/2) \rfloor + 1), \cdots, (n-3), (n-2), (n-1)$
  - For even n, each value is occurring twice. For odd n we include an extra value of, $\lfloor (n/2) \rfloor$
  - Since q is selected from an array of n elements, $\Pr(q \in \{1, 2, \cdots n\}) = 1/n$
- Now do some routine algebra to establish an upper bound for $E(T(n))$

$$E(T(n)) \leq E[(1/n)(T(n-1) + (\sum_{k=2}^{n-2}] T(n-k-1)) + T(n-1)) + O(n)]$$
$$\leq E[(1/n)(\sum_{k=1}^{n-1}] T(n-k))] + O(n)$$
$$= (1/n) \sum_{k=1}^{n-1} E[T(n-k-1)] + O(n)$$

- Since each term is occurring twice and minimum value of $(n-k) = \lfloor (n/2) \rfloor$, we can write the inequality as,
- $E(T(n) \leq \frac{2}{n} \sum_{k=\lfloor (n/2) \rfloor}^{n-1} E[T(n-k)] + O(n)$
- Using substitution we can show that $E[T(n)] = O(n)$
- Detailed algebra is shown in the textbook
- $E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor (n/2) \rfloor}^{n-1} ck + an$

$$\leq \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor (n/2) \rfloor - 1} k \right) + an$$
$$= \frac{2c}{n} \left(\frac{n^2-n}{2} - \frac{(n^2/4 - 3n/2 + 2)}{2} \right) + an$$
$$= \frac{c}{n} \left(\frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an$$
$$= c \left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an$$
$$\leq \frac{3cn}{4} + \frac{c}{2} + an$$
$$= cn - \frac{cn}{4} + \frac{c}{2} + an$$
$$= cn - \left(\frac{cn}{4} - \frac{c}{2} - an \right)$$

- Now we need to find values of c and n such that the induction
- It can be shown that for $c > 4a$ and $n \geq \frac{2c}{c-4a}$, $E[T(n)] = O(n)$

Computation of Expected Execution Time of An Algorithm

- Math foundation
  - $E(X) = \sum_{i=1}^{n} E(X_i)$ for $X = X_1 + X_2 + \cdots + X_n$.
  - For an indicator random variable X_A of an event A, $E(X_A) = \text{Prob.}(A)$.
- Steps for estimation of execution time
  - **Identify An Event**:
    - For hiring: A candidate is hired
    - For Quicksort: X_i is compared with X_j
    - For Selection: After randomized partition, the pivot is the desired element.
  - Express execution time as sum of execution times of the identified event
  - Do necessary algebra and approximations
- Question?